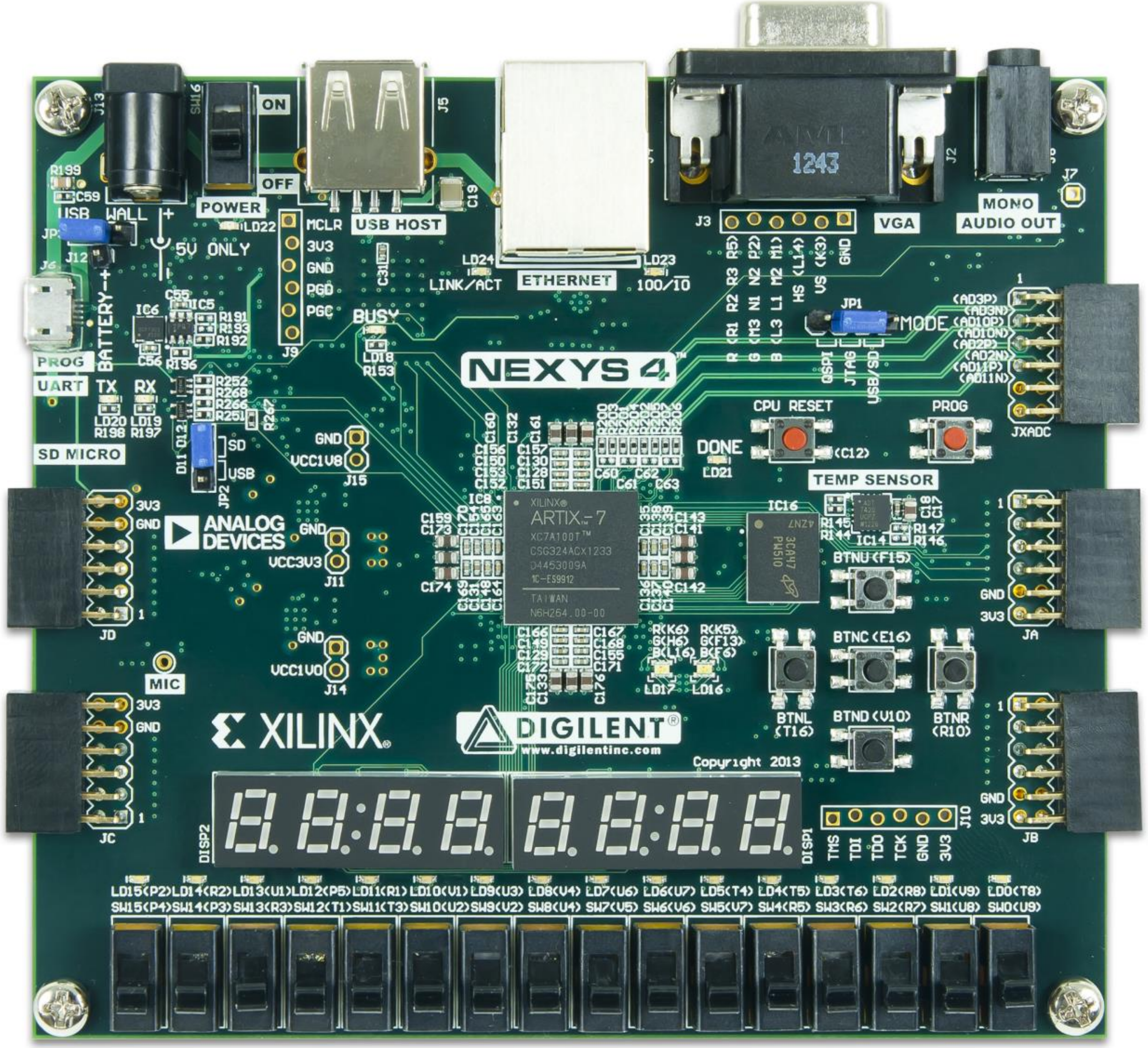
Final Project CAN Controller

Developed by: David Gouin, William Courtioux and Garrett Willobee

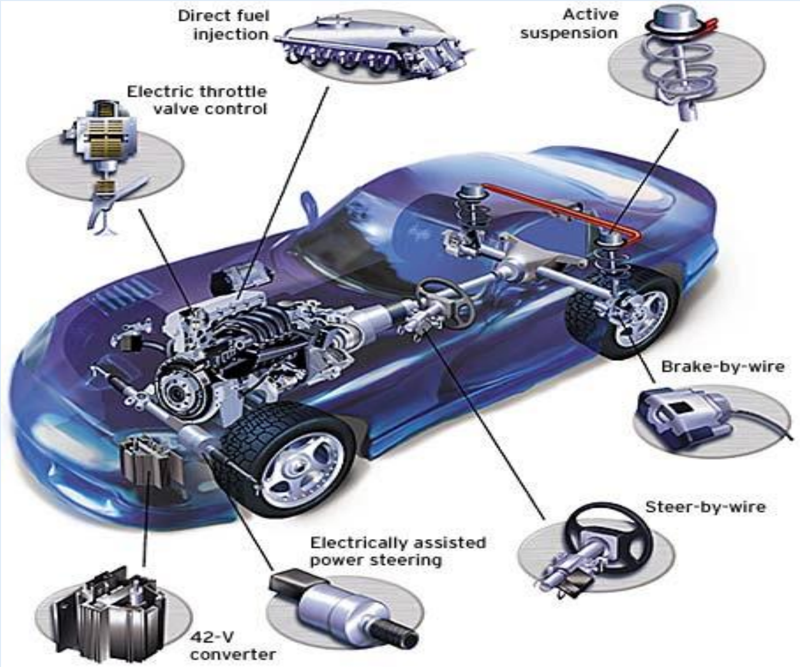
Nexys 4

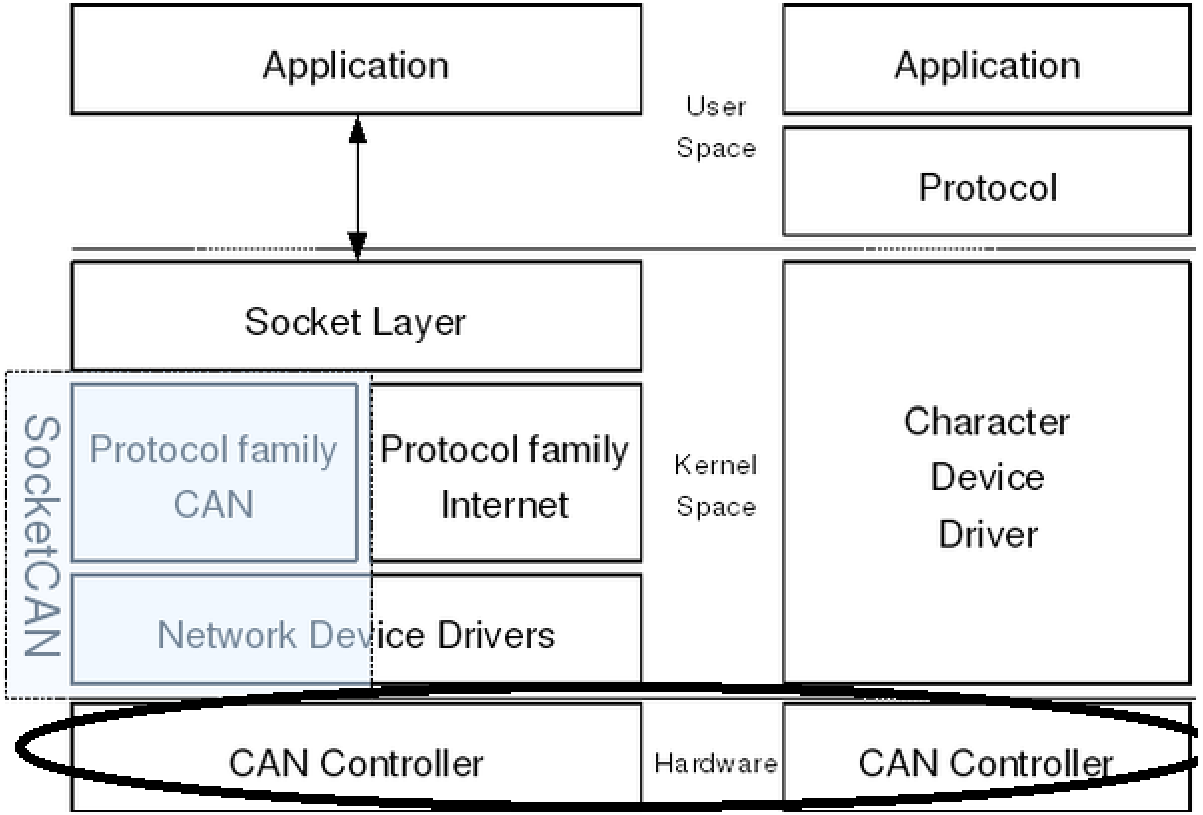


NEXYS 4
XILINX
ARTIX-7
XC7A100T
DIGILENT
www.digilentinc.com
Copyright 2013
ANALOG DEVICES
ON
OFF
POWER
USB HOST
ETHERNET
VGA
MONO AUDIO OUT
LINK/ACT
100/10
5V ONLY
WALL
USB
BATTERY
PROG
UART
TX
RX
SD MICRO
MIC
BUSY
DONE
CPU RESET
TEMP SENSOR
MODE
QSPI
JTAG
USB/SD
BTNU (F15)
BTNC (E16)
BTNL (T16)
BTND (V10)
BTNR (R10)
JXADC
JA
JB
JC
JD
DISP1
DISP2
LD15(P2) LD14(R2) LD13(U1) LD12(P5) LD11(R1) LD10(V1) LD9(U3) LD8(V4) LD7(U6) LD6(V7) LD5(T4) LD4(T5) LD3(T6) LD2(R8) LD1(V9) LD0(T8)
SW15(P4) SW14(P3) SW13(R3) SW12(T1) SW11(T3) SW10(U2) SW9(V2) SW8(U4) SW7(V5) SW6(V6) SW5(V7) SW4(R5) SW3(R6) SW2(R7) SW1(U8) SW0(U9)
TMS TDI TDO TCK GND 3V3

Understanding The Scope Of The Project

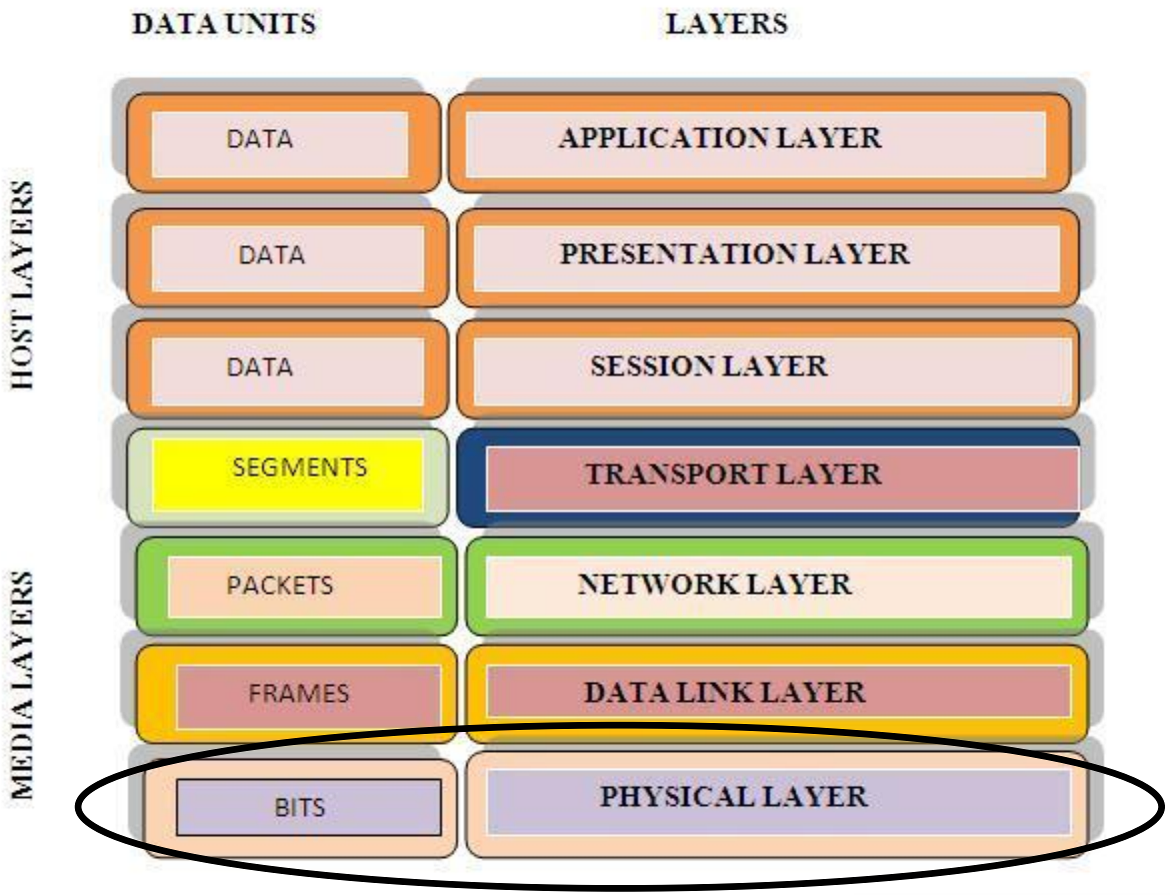
Where is CAN used?





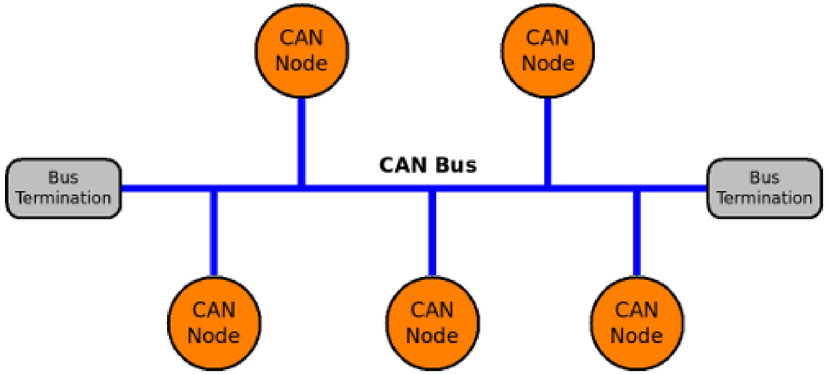
WHAT WE DESIGNED

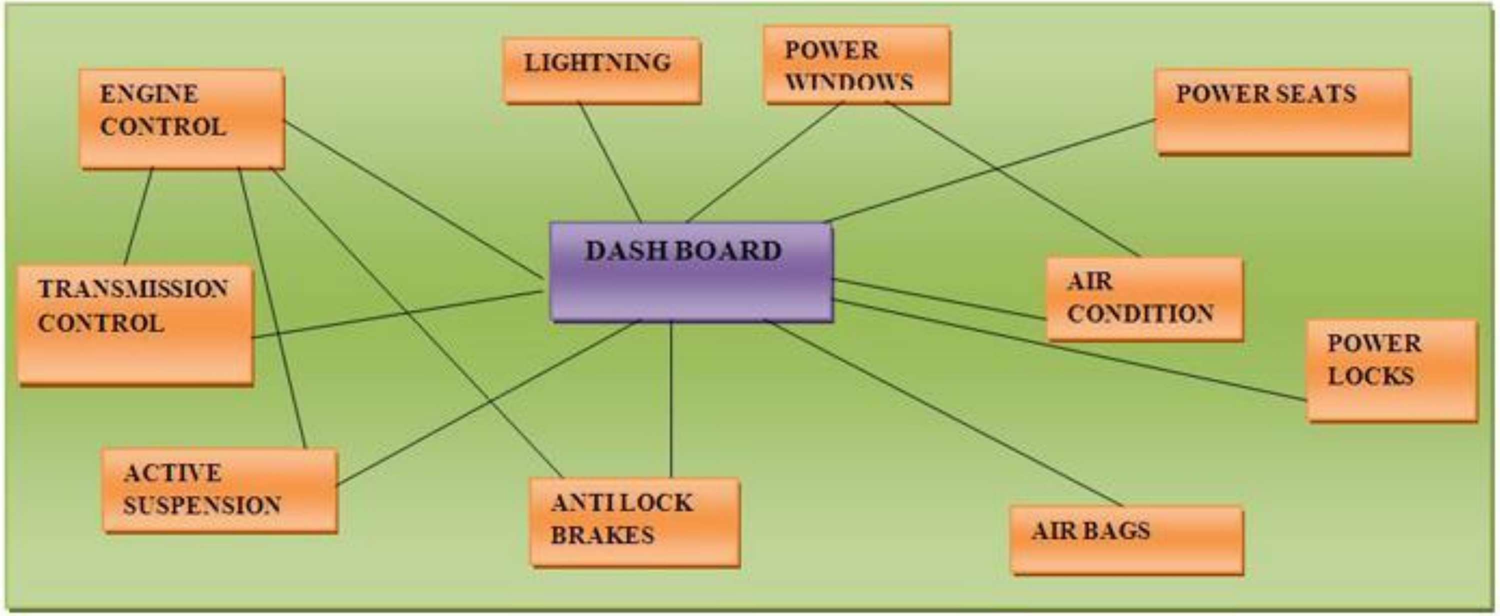
A BETTER IDEA

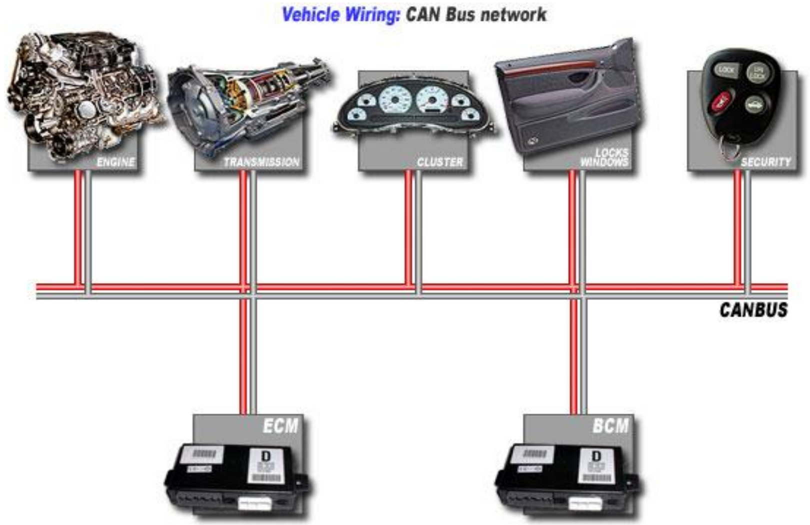


	DATA UNITS	LAYERS
HOST LAYERS	DATA	APPLICATION LAYER
	DATA	PRESENTATION LAYER
	DATA	SESSION LAYER
	SEGMENTS	TRANSPORT LAYER
MEDIA LAYERS	PACKETS	NETWORK LAYER
	FRAMES	DATA LINK LAYER
	BITS	PHYSICAL LAYER

APPLICATION

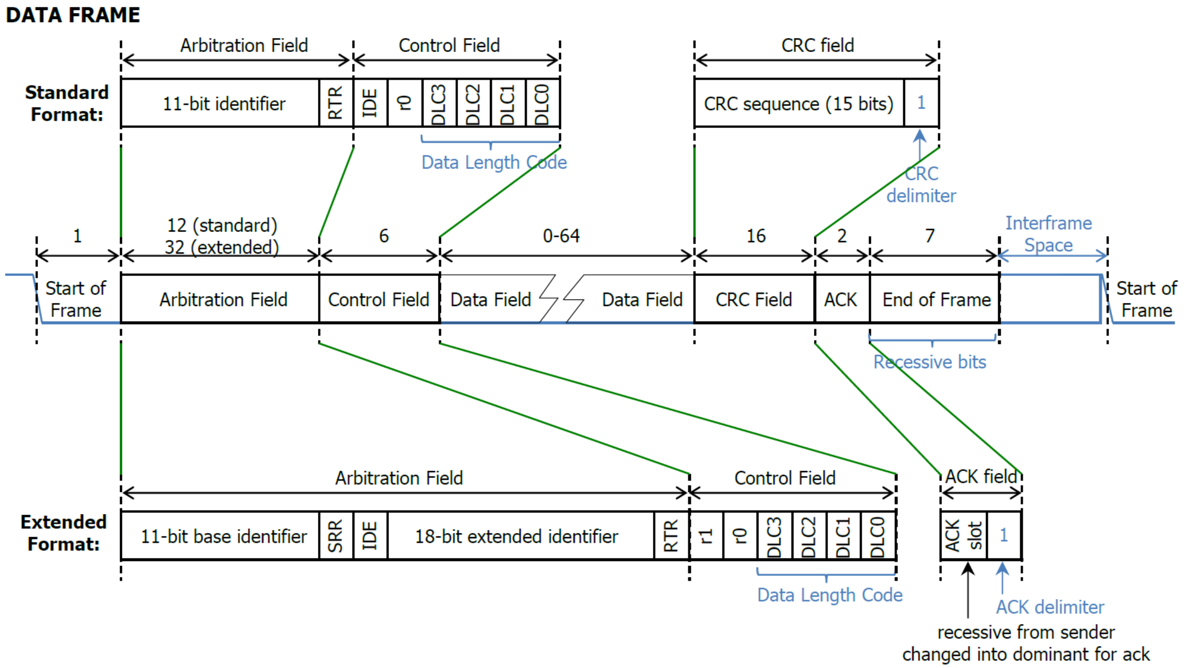


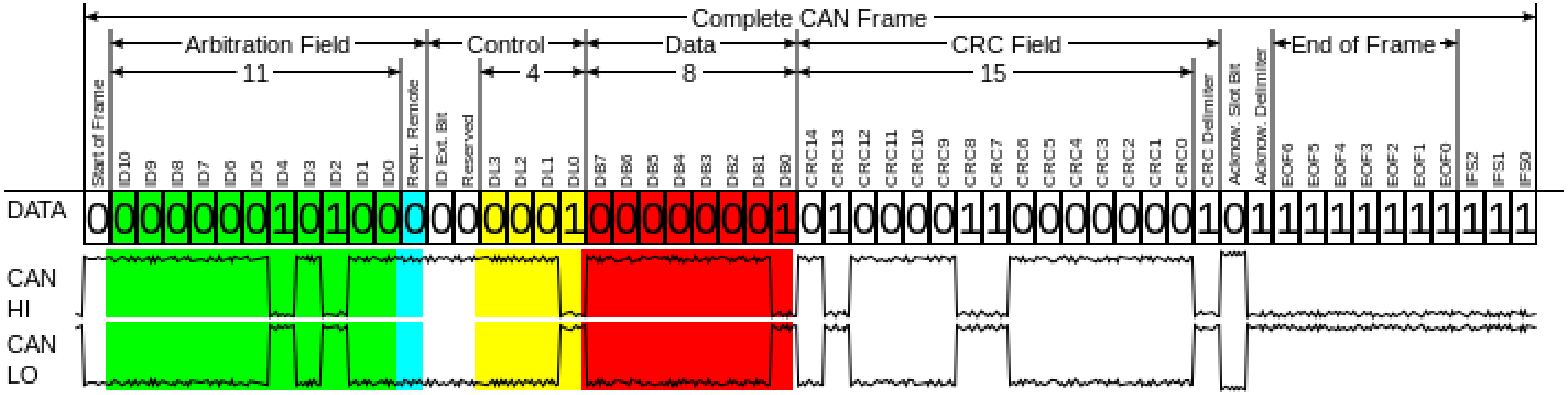




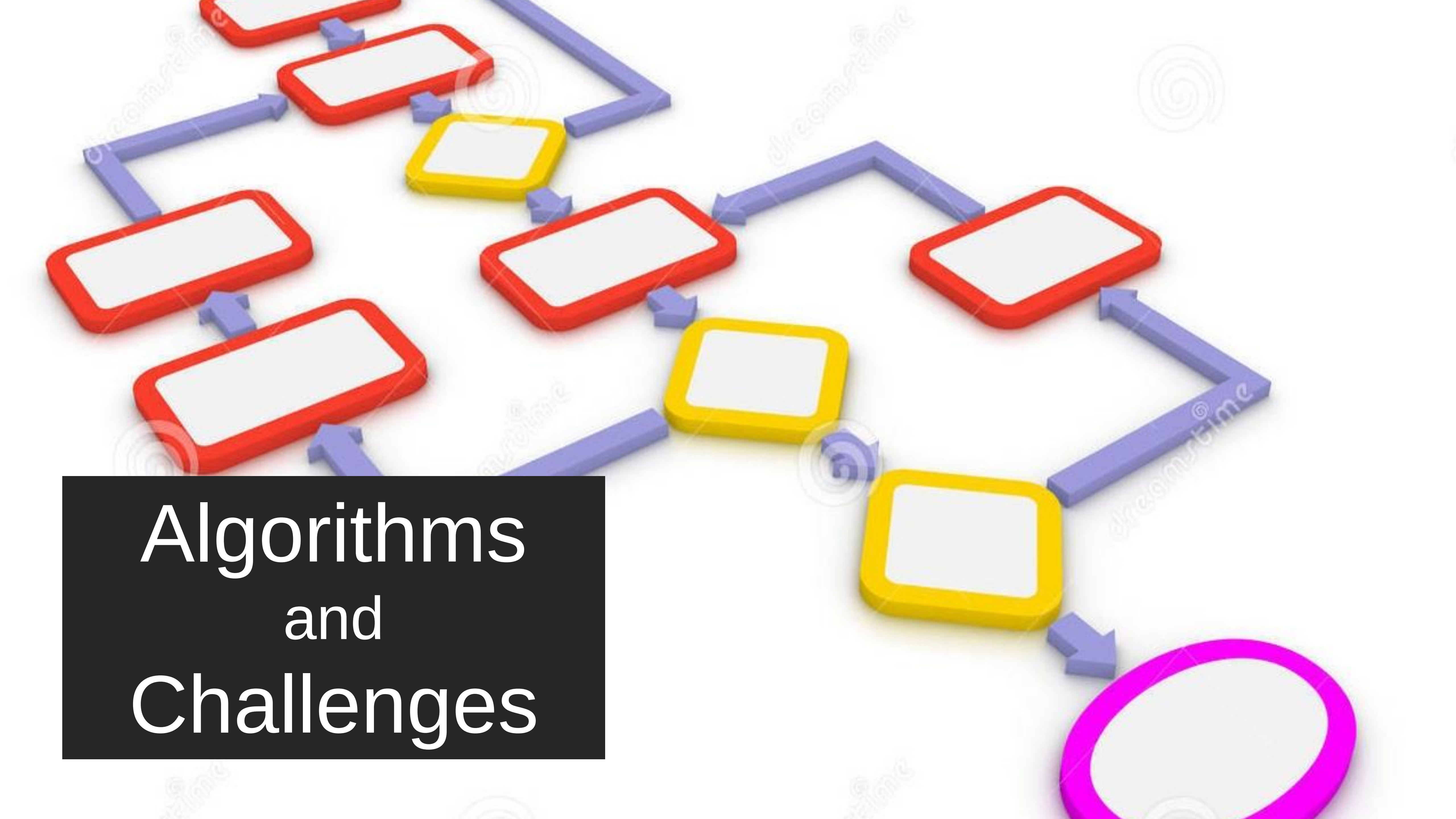
Function Goals:

- Detection of start bit
- Detection of stuffed bits
- Properly recording each field
- Checking CRC
- Sending a reply message based of incoming data
- Creating a reply message with a generated CRC value and stuffed bits
- Displaying data recorded

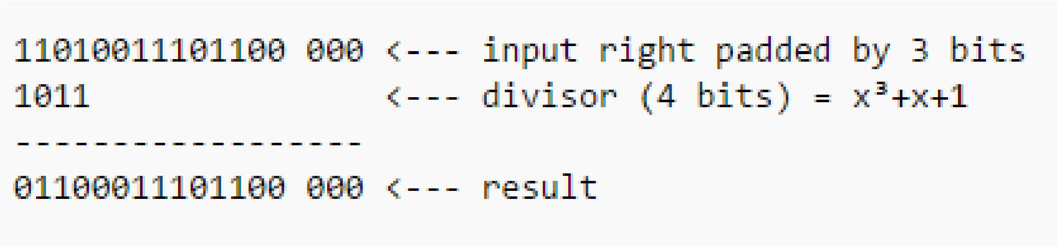




FINALLY! Our Project



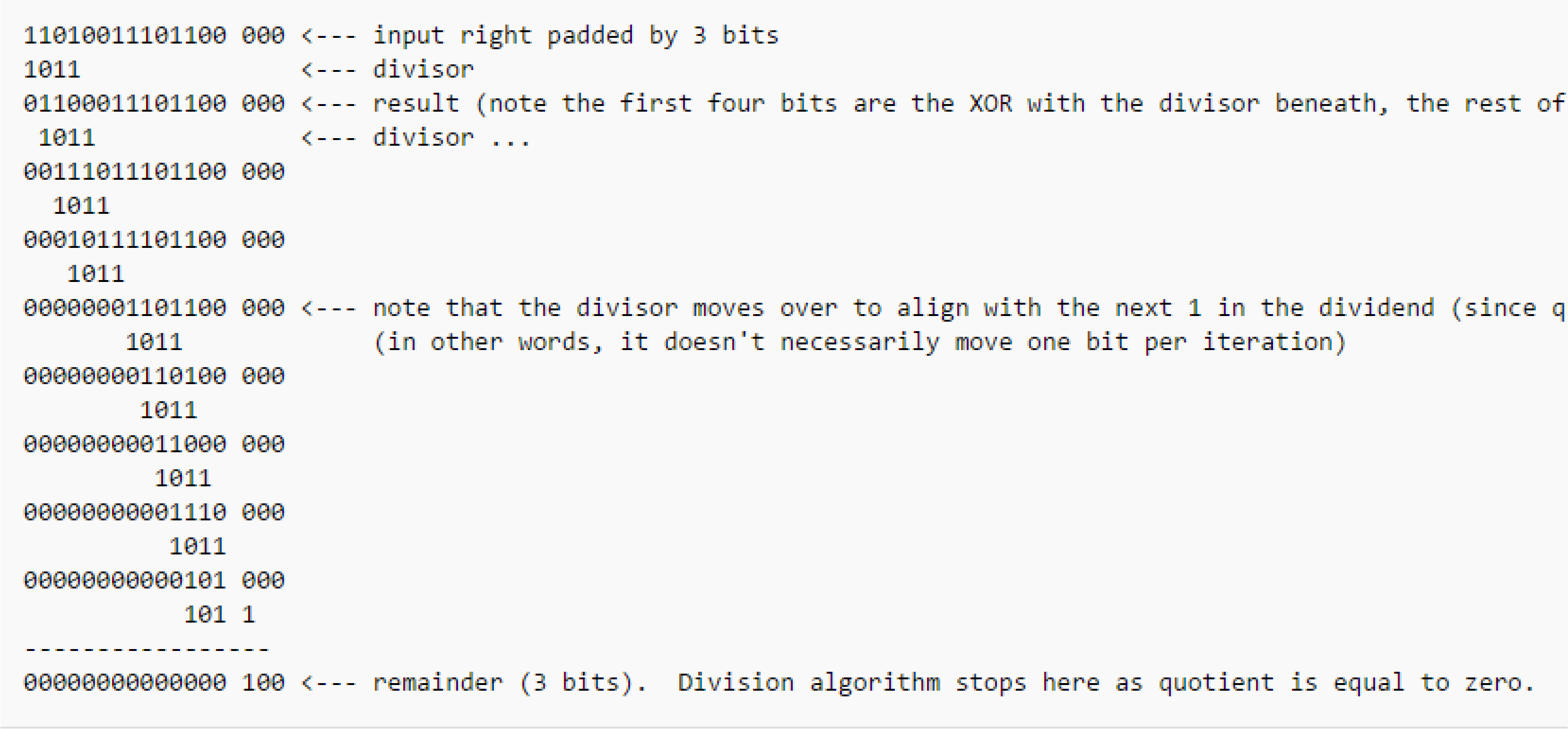
Algorithms
and
Challenges



```
11010011101100 000 <--- input right padded by 3 bits
1011               <--- divisor (4 bits) = x³+x+1
------------------
01100011101100 000 <--- result
```

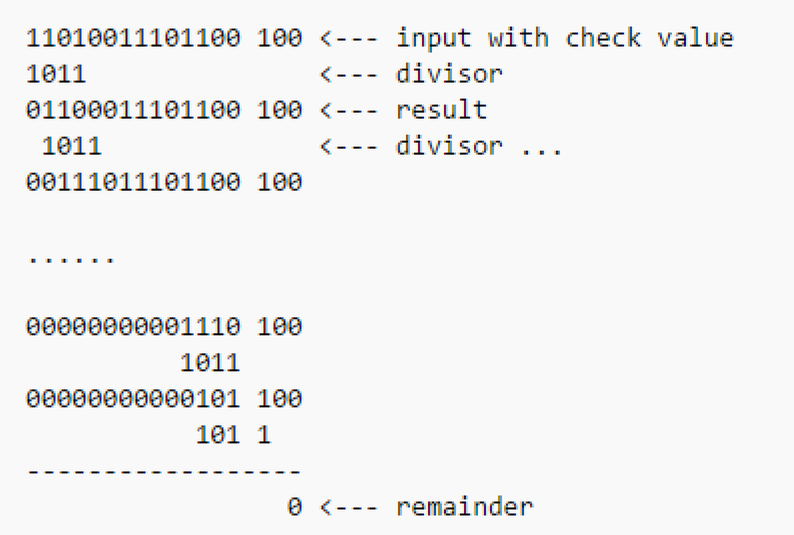
First

Second



```
11010011101100 000 <--- input right padded by 3 bits
1011               <--- divisor
01100011101100 000 <--- result (note the first four bits are the XOR with the divisor beneath, the rest of the bits are unchanged)
 1011              <--- divisor ...
00111011101100 000
  1011
00010111101100 000
   1011
00000001101100 000 <--- note that the divisor moves over to align with the next 1 in the dividend (since quotient for that step was zero)
       1011             (in other words, it doesn't necessarily move one bit per iteration)
00000000110100 000
        1011
00000000011000 000
         1011
00000000001110 000
          1011
00000000000101 000
           101 1
-----------------
00000000000000 100 <--- remainder (3 bits).  Division algorithm stops here as quotient is equal to zero.
```

Last



```
11010011101100 100 <--- input with check value
1011               <--- divisor
01100011101100 100 <--- result
 1011              <--- divisor ...
00111011101100 100

......

00000000001110 100
          1011
00000000000101 100
           101 1
-----------------
                0 <--- remainder
```

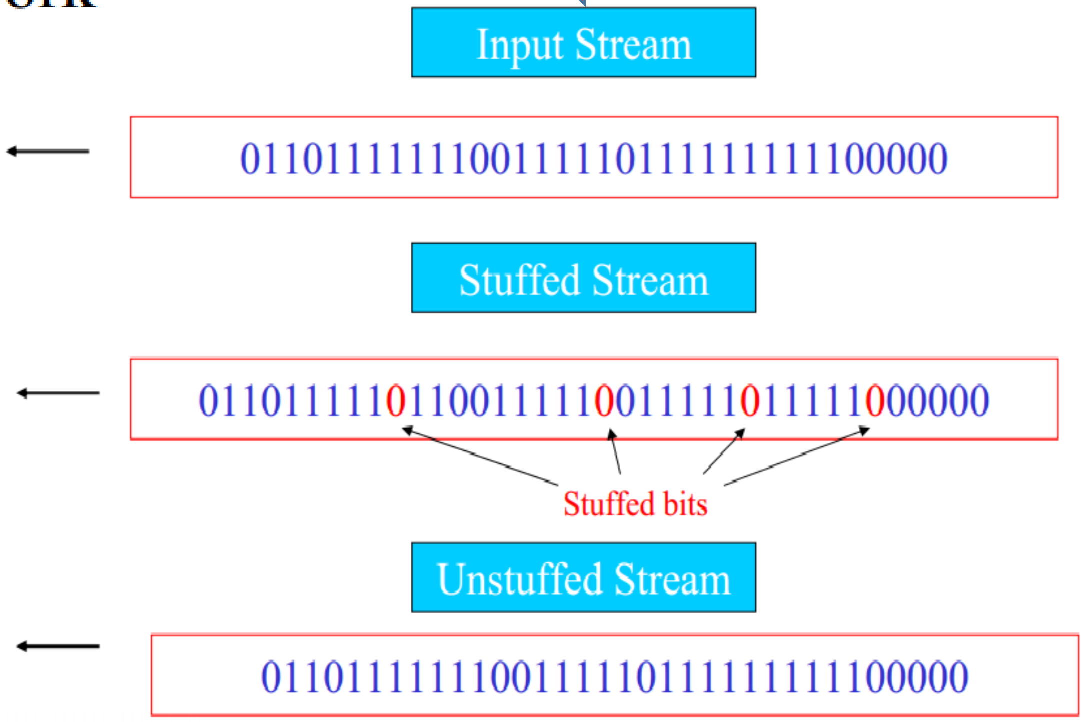
CRC-3 ALGORITHM EXAMPLE

Whenever sender data link layer encounters *five consecutive ones* in the data stream, it automatically stuffs a 0 bit into the outgoing stream.

When the receiver sees *five consecutive incoming ones followed by a 0 bit*, it automatically destuffs the 0 bit before sending the data to the network layer.

General Idea

BIT STUFFING EXAMPLE



Input Stream

01101111111001111101111111111100000

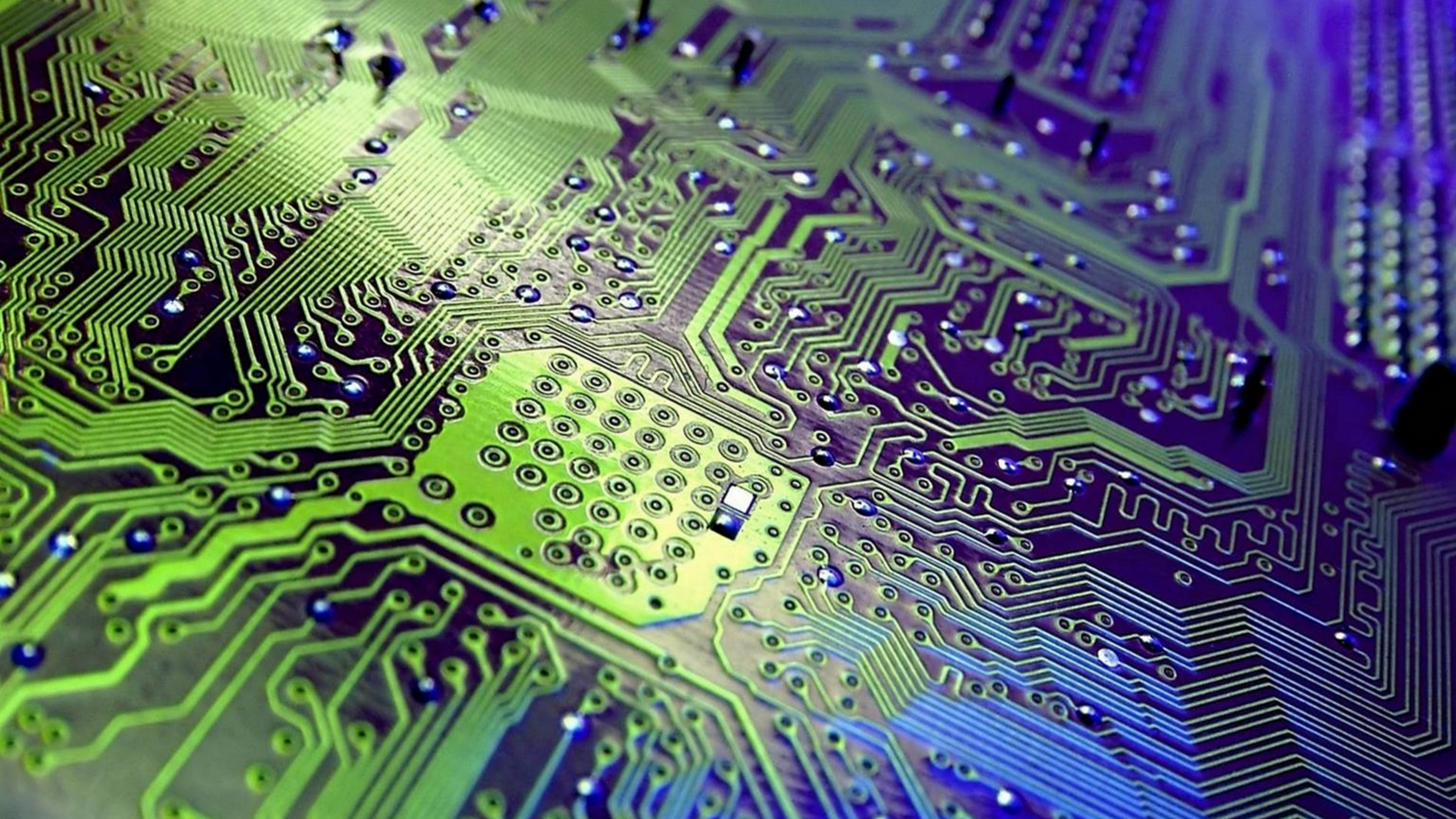
Stuffed Stream

01101111101100111110011111011111000000

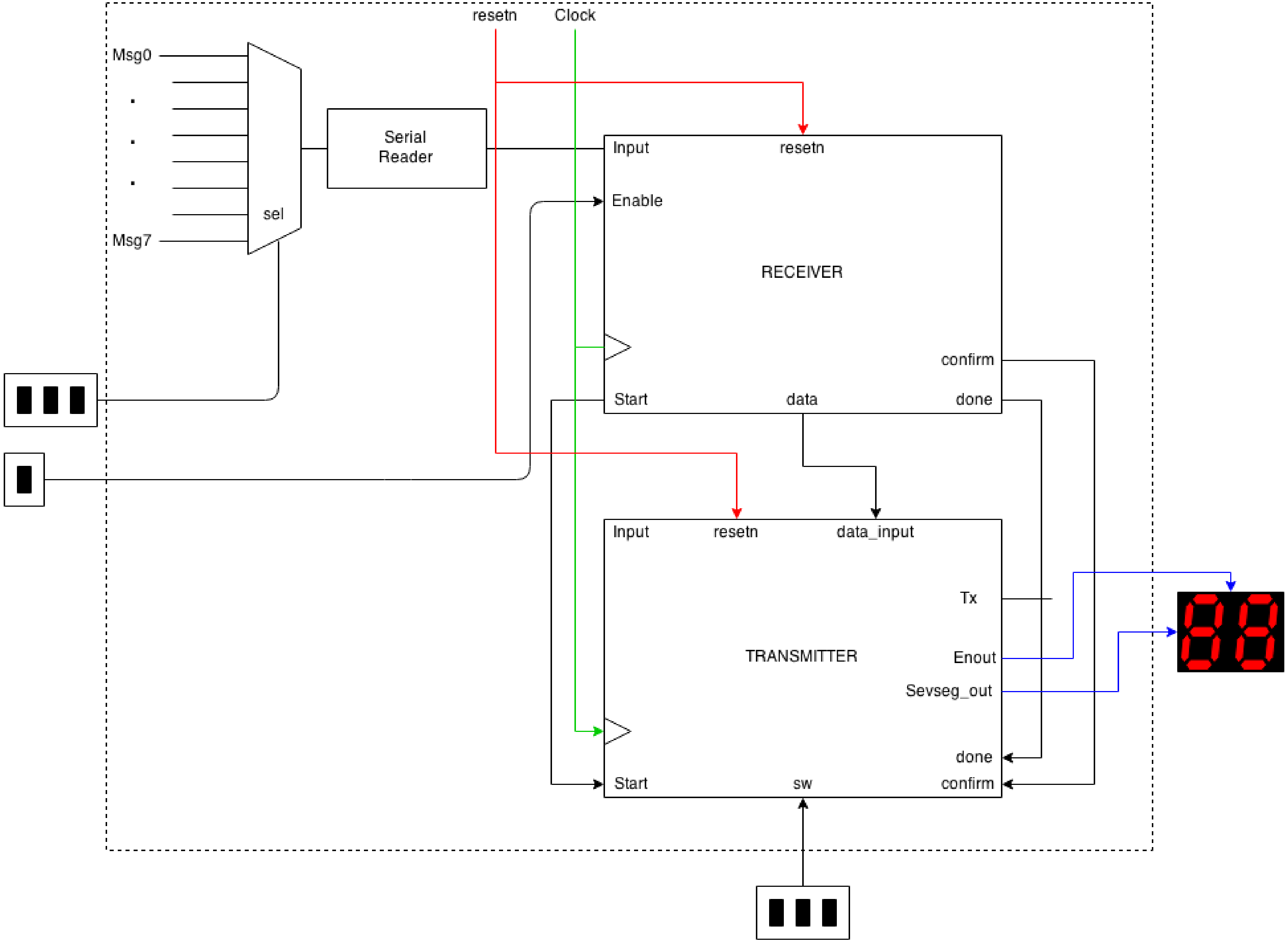
Stuffed bits

Unstuffed Stream

01101111111001111101111111111100000

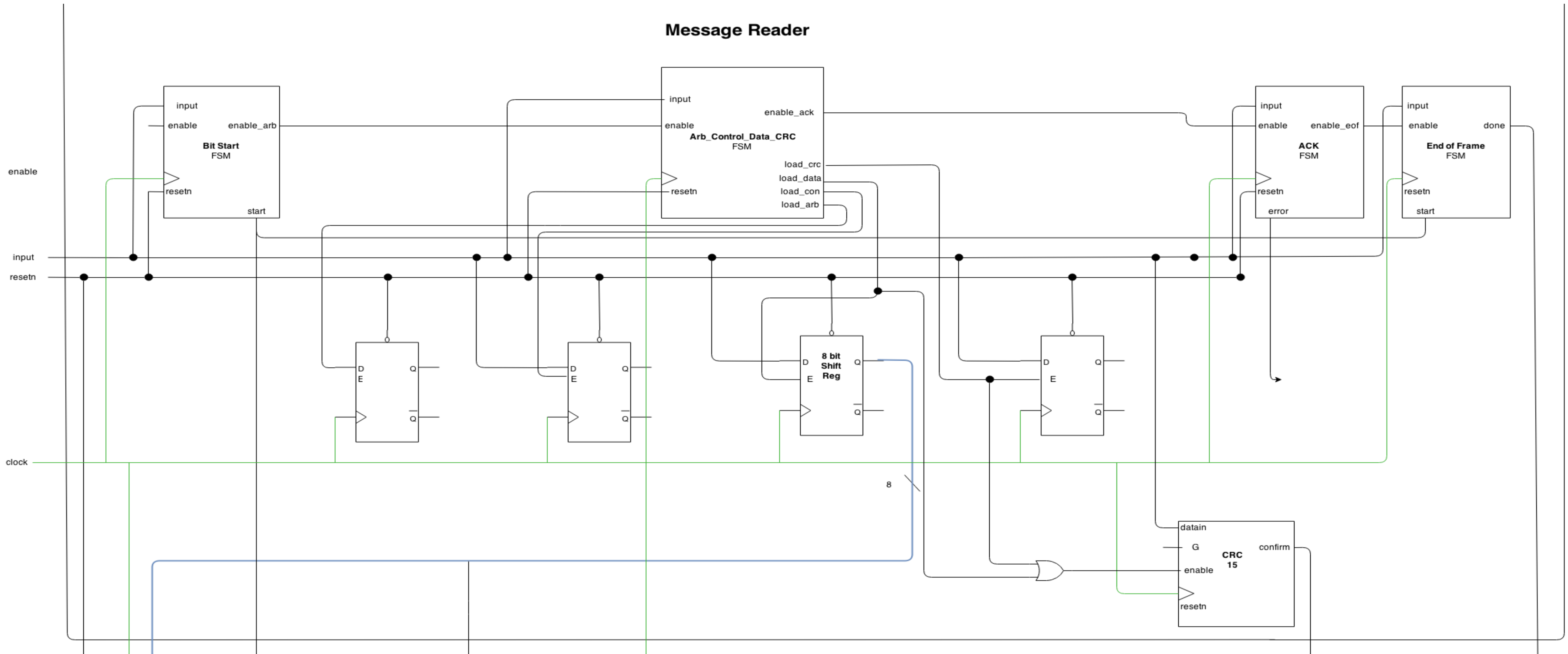


Our Final TOP LEVEL

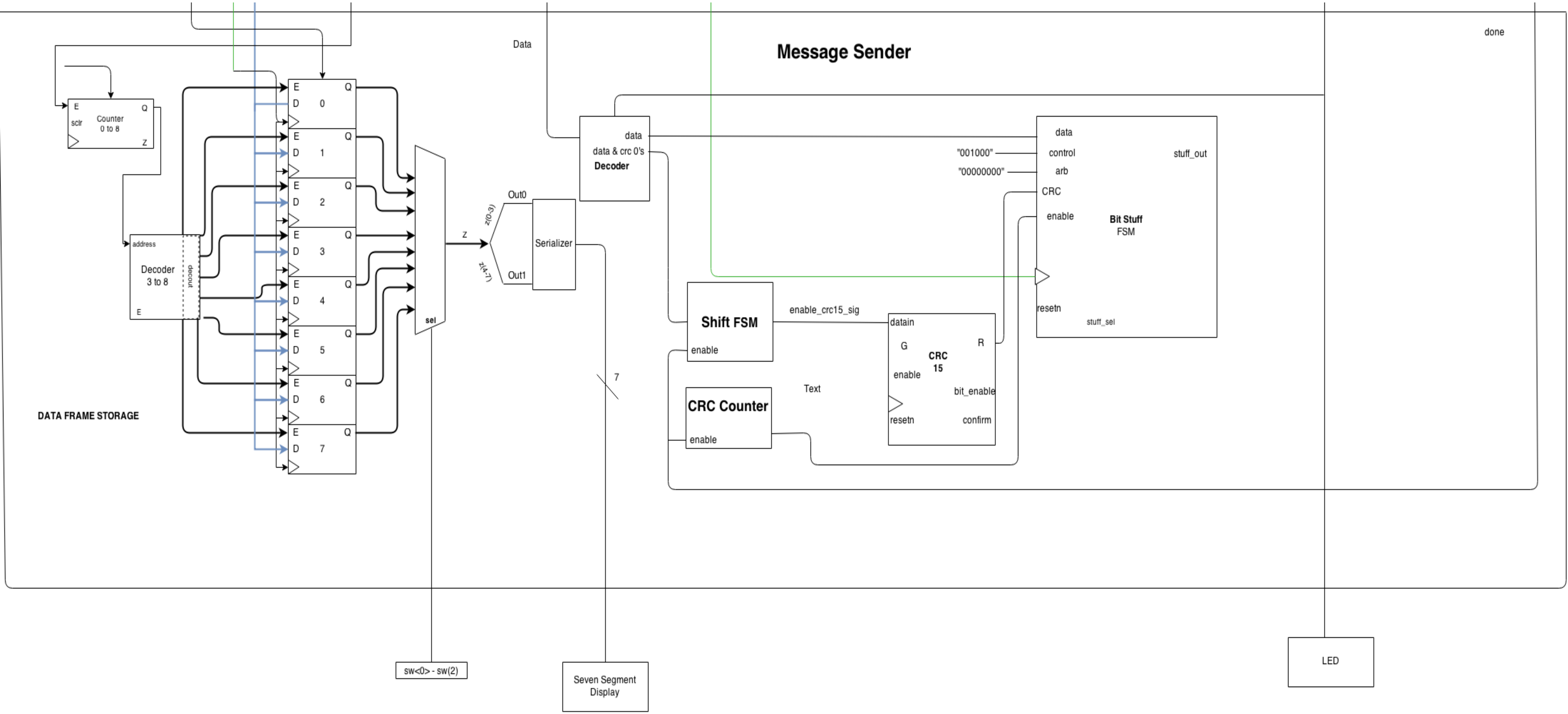


Custom Diagrams From Our Perspective

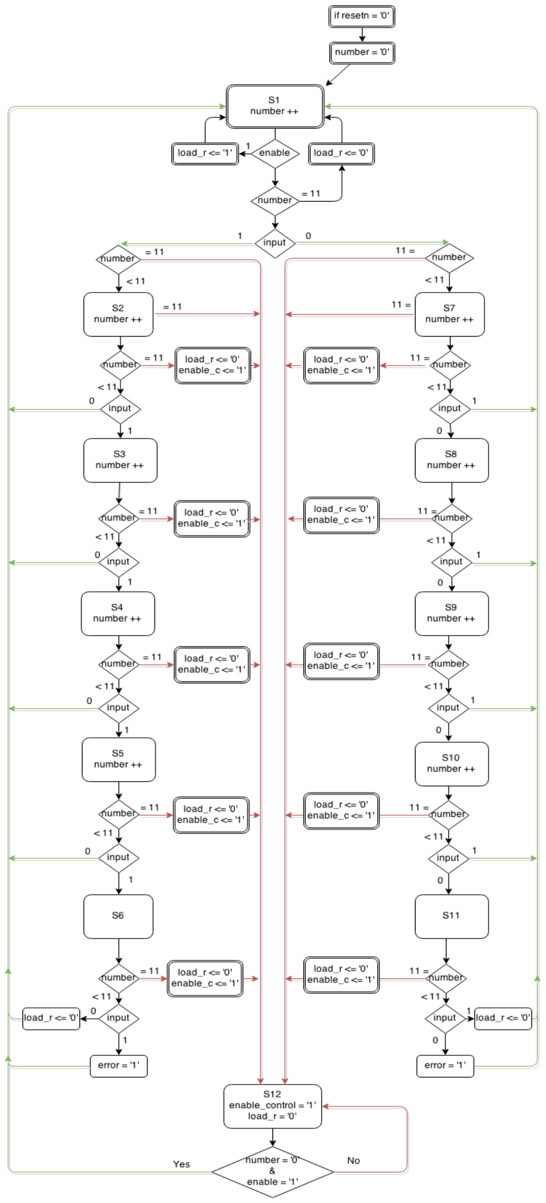
Part (1) Message Reader

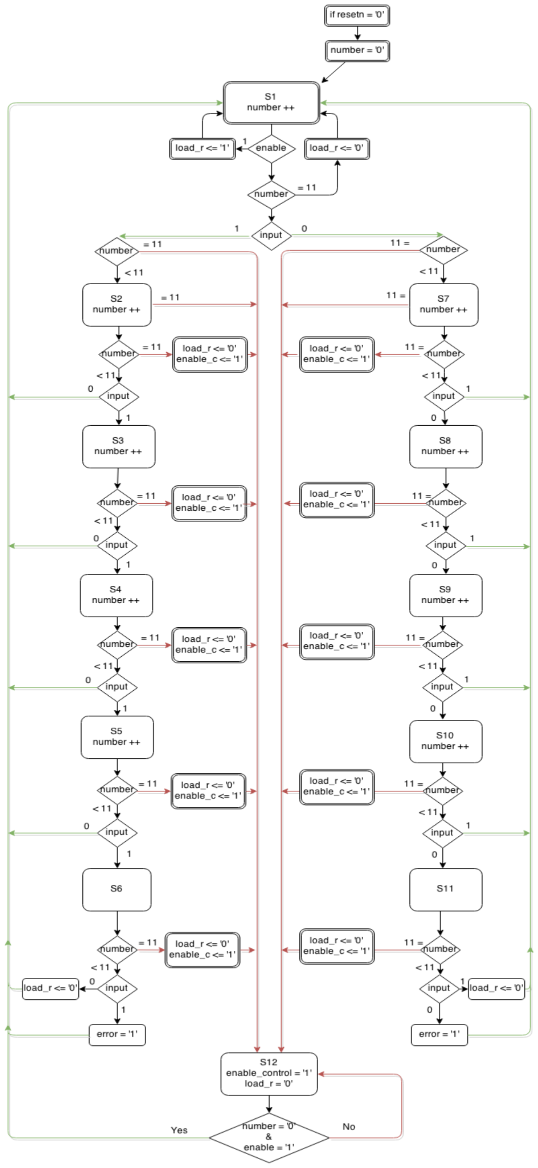


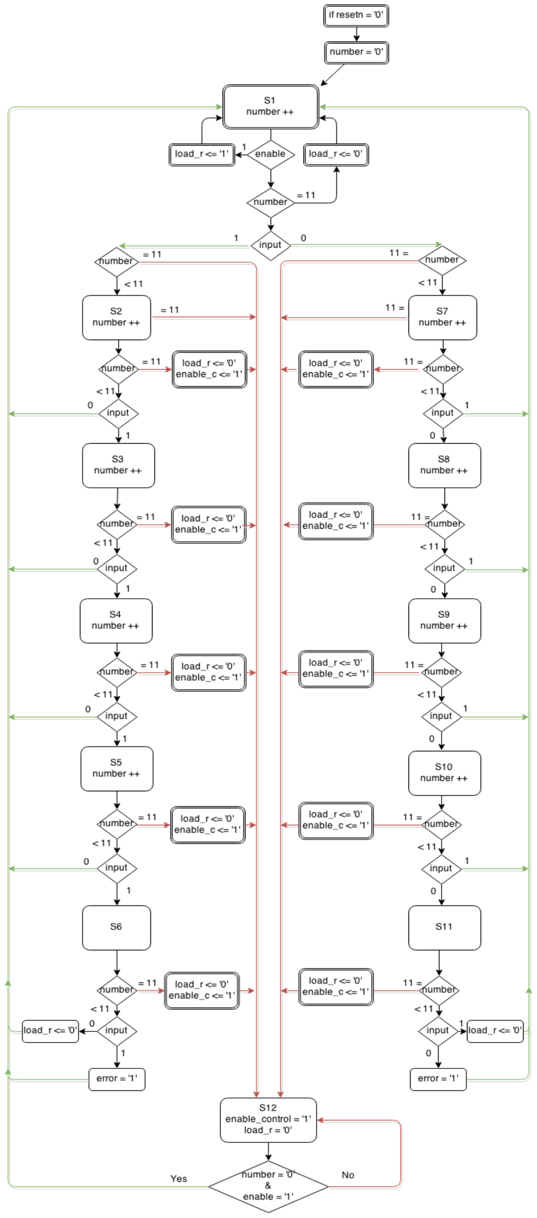
Part (2) Message Sender



Creating the FSM's





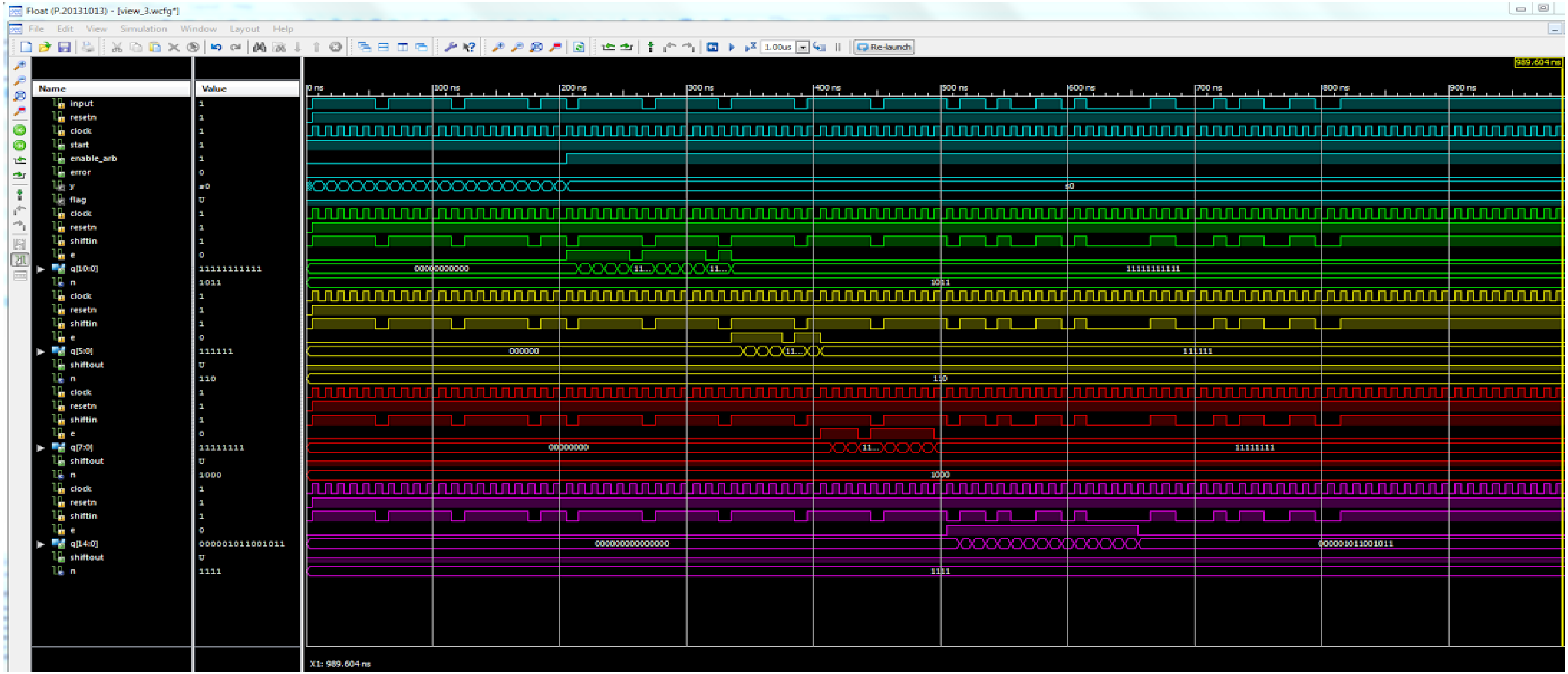


14 Total FSM’s!

DEBUGGING

TOP READ

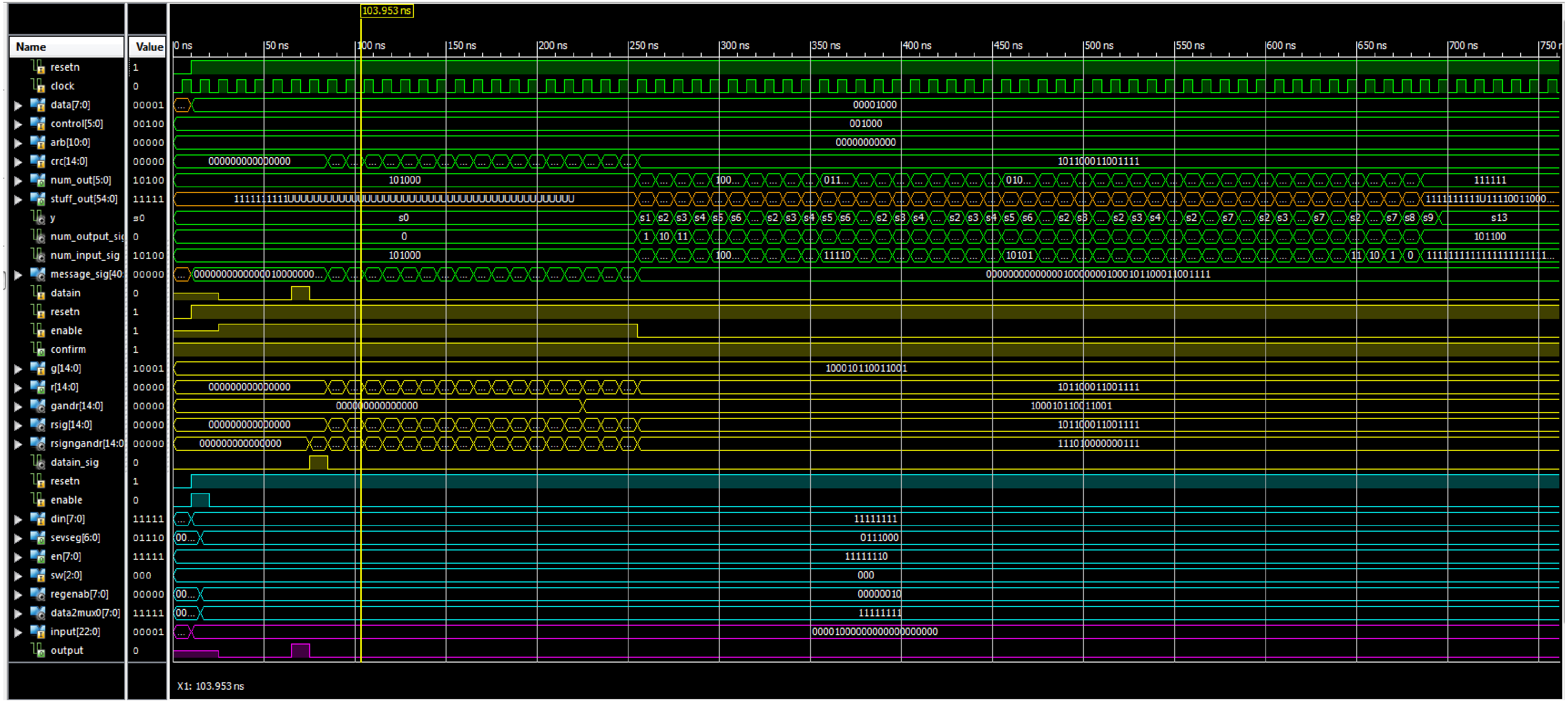
Verifying all the fields have been recorded properly



Name	Value
input	1
resetn	1
clock	1
start	1
enable_arb	1
error	0
y	a0
flag	U
clock	1
resetn	1
shiftin	1
e	0
q[10:0]	11111111111
n	1011
clock	1
resetn	1
shiftin	1
e	0
q[5:0]	111111
shiftout	U
n	110
clock	1
resetn	1
shiftin	1
e	0
q[7:0]	11111111
shiftout	U
n	1000
clock	1
resetn	1
shiftin	1
e	0
q[14:0]	000001011001011
shiftout	U
n	1111

Top Send

Verifying data sent, CRC, and proper bit stuffing



Name	Value
resetn	1
clock	0
data[7:0]	00001
control[5:0]	00100
arb[10:0]	00000
crc[14:0]	00000
num_out[5:0]	10100
stuff_out[54:0]	11111
y	s0
num_output_sig	0
num_input_sig	10100
message_sig[40:	00000
datain	0
resetn	1
enable	1
confirm	1
g[14:0]	10001
r[14:0]	00000
gandr[14:0]	00000
rsig[14:0]	00000
rsigngandr[14:0	00000
datain_sig	0
resetn	1
enable	0
din[7:0]	11111
sevseg[6:0]	01110
en[7:0]	11111
sw[2:0]	000
regenab[7:0]	00000
data2mux0[7:0]	11111
input[22:0]	00001
output	0

Completed functionality:

- Detect a message and store the data
- Generate a message with CRC and stuffed bits

Unfinished functionality:

- Synchronization of nodes
- Output message serially

THE END!!!